```
********************************************************************************

business_request_models.go_a8061d3d36d0887e9eeb536aeb66001e.txt

********************************************************************************

package request_models

type BusinessRequest struct {
Name        string `json:"business_name"`
Address     string `json:"business_address"`
TotalTables int    `json:"total_tables"`
OpeningTime string `json:"opening_time"`
ClosingTime string `json:"closing_time"`
}

********************************************************************************

********************************************************************************

cancel_order.go_7dbe166a4c3041fbbb3a07ed0519813c.txt

********************************************************************************

package orders

import (
"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_orders"
"github.com/jimzord12/serve-tech/api/models"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// CancelOrder - DELETE handler
func CancelOrder(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

orderID := c.Params("id")
order, err := db_orders.CancelOrderInDatabase(db, orderID)

if err != nil {
utils.LogErrorToConsole("Handler: 'CancelOrder'", err)
return c.Status(fiber.StatusNotFound).JSON(fiber.Map{"status": "error", "message": "Order not found or c
}

if order.Status == models.OrderCancelled {
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Order is already canceled",
"data": nil,
})
```

```
} else if order.Status == models.OrderCompleted {
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Order is already completed and cannot be canceled",
"data": nil,
})
}

return c.JSON(fiber.Map{"status": "success", "message": "Order canceled", "data": order})
}
```

****

****

complete_order.go_8af5f70df1f42de796ec96524f04ad1e.txt

****

```
package orders

import (
"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_orders"
"github.com/jimzord12/serve-tech/api/models"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// CompleteOrder - PUT handler
func CompleteOrder(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

orderID := c.Params("id")
order, err := db_orders.CompleteOrderInDatabase(db, orderID)
if err != nil {
utils.LogErrorToConsole("Handler: 'CompleteOrder'", err)
return c.Status(fiber.StatusNotFound).JSON(fiber.Map{"status": "error", "message": "Order not found or c
}

if order.Status == models.OrderCompleted {
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Order is already completed",
"data": nil,
})
} else if order.Status == models.OrderCancelled {
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Order is already canceled and cannot be completed",
"data": nil,
})
```

```
}

return c.JSON(fiber.Map{"status": "success", "message": "Order completed", "data": order})
}

********************************************************************************

********************************************************************************

create_customer.go_89b6cb06073e02cf071e23314befca59.txt

********************************************************************************

package customers

import (
"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/models"
"github.com/jimzord12/serve-tech/api/models/request_models"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// CreateCustomer - POST handler
func CreateCustomer(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

// Instance to store the incoming customer data
customerRequest := new(request_models.CreateCustomerRequest)

// Parse the JSON body into the customerRequest instance
if err := c.BodyParser(customerRequest); err != nil {
utils.LogErrorToConsole("Customer Handlers::Cannot parse JSON", err)
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Cannot parse JSON",
"data": err.Error(),
})
}

// Create a Customer model from the request data
customer := models.Customer{
FirstName: customerRequest.FirstName,
LastName:  customerRequest.LastName,
Email:     customerRequest.Email,
Phone:     customerRequest.Phone,
}

// Save the new customer to the database
if err := db.Create(&customer).Error; err != nil {
utils.LogErrorToConsole("Customer Handlers::Customer Creation Failed!", err)
return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
```

```
"status": "error",
"message": "Could not create customer",
"data": err.Error(),
})
}

// Return the newly created customer
return c.Status(fiber.StatusCreated).JSON(fiber.Map{
"status": "success",
"message": "Customer created",
"data": customer,
})
}
```

****

****

create_history_order.go_7bf0fae6d07d6ac1bb7be8eaad237a90.txt

****

```
package history_orders

import (
"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_history_orders"
"github.com/jimzord12/serve-tech/api/models/request_models"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// CreateHistoryOrder - POST handler
func CreateHistoryOrder(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

historyOrderRequest := new(request_models.CreateHistoryOrderRequest) // This is a Pointer
if err := c.BodyParser(historyOrderRequest); err != nil {
utils.LogErrorToConsole("Handler: 'CreateHistoryOrder', Cannot parse JSON", err)
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Cannot parse JSON",
"data": err.Error(),
})
}

historyOrder, err := db_history_orders.CreateHistoryOrderInDatabase(db, historyOrderRequest)
if err != nil {
utils.LogErrorToConsole("Handler: 'CreateHistoryOrder'", err)
return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
"status": "error",
```

```
"message": "Failed to create history order",
"data": err.Error(),
})
}

return c.JSON(fiber.Map{
"status": "success",
"message": "History order created",
"data": historyOrder,
})
}
```

****

****

create_order.go_f57a4cc30f6c6df52fbf53571cc768be.txt

****

```
package orders

import (
"strings"

"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_orders"
"github.com/jimzord12/serve-tech/api/db_functions/db_utils"
"github.com/jimzord12/serve-tech/api/models/request_models"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// CreateOrderRequest - POST handler
// This Action can be performed by a Customer, Waiter, or Adminw
func CreateOrder(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

orderRequest := new(
request_models.CreateOrderRequest,
)
if err := c.BodyParser(orderRequest); err != nil {
utils.LogErrorToConsole("Handler: 'CreateOrder', Cannot parse JSON", err)
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Cannot parse JSON",
"data": err.Error(),
})
}

////// ■ CHECKING TABLE ■ //////
```

```
isTableAvailable, err := db_utils.IsTableAvailable(db, orderRequest.TableID)
if err != nil {
if err.Error() == "table does not exist in db" { // ■ [■]
utils.LogErrorToConsole("Handler: 'CreateOrder', Table does NOT exist in DB", err)
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Table does not exist in db",
"data": nil,
})
} else if strings.Contains(err.Error(), "is already occupied with order ID") { // ■ [■]
utils.LogErrorToConsole("Handler: 'CreateOrder', Table is already occupied with an Order", err)
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": err.Error(),
"data": nil,
})
} else { // ■ [■]
utils.LogErrorToConsole("Handler: 'CreateOrder', Table Availability check failed", err)
return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
"status": "error",
"message": "Table Availability check failed",
"data": err.Error(),
})
}
}
if !isTableAvailable { // ■ [■]
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Table's CurrentOrderID in NULL but its Status is NOT Available. This should not happen. Ple
"data": nil,
})
}

////// ■ CHECKING CUSTOMER ■ //////
customerExists, err := db_utils.DoesCustomerExist(db, orderRequest.CustomerID)
if err != nil { // ■ [■]
utils.LogErrorToConsole("Handler: 'CreateOrder', Customer check failed", err)
return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
"status": "error",
"message": "Customer check failed",
"data": err.Error(),
})
}
if !customerExists { // ■ [■]
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Customer does not exist",
"data": nil,
})
}

////// ■ CHECKING DATE ■ //////
if !utils.IsDateFormatValid(orderRequest.OrderTime) { // ■ [■]
```

```
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": "Invalid date format, should be 'YYYY-MM-DD HH:MM:SS'",
"data": nil,
})
}

////// ■ CREATING ORDER ■ //////
order, err := db_orders.CreateOrderInDatabase(db, orderRequest)
if err != nil {
utils.LogErrorToConsole("Handler: 'CreateOrder'", err)
return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
"status": "error",
"message": err.Error(),
"data": nil,
})
}

return c.Status(fiber.StatusCreated).JSON(fiber.Map{
"status": "success",
"message": "Order created",
"data": order,
})
}
```

****

****

create_tables.go_62ce5933a8ace0264a29056c3b4286b7.txt

****

```
package tables

import (
"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_tables"
"github.com/jimzord12/serve-tech/api/models/request_models"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// CreateTable - POST handler
func CreateTable(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

tableRequest := new(request_models.CreateTableRequest)
if err := c.BodyParser(tableRequest); err != nil {
utils.LogErrorToConsole("Could not parse JSON", err)
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
```

```
"message": "Cannot parse JSON",
"data": err.Error(),
})
}

table, err := db_tables.CreateTableInDatabase(db, tableRequest)
if err != nil {
utils.LogErrorToConsole("Could not create table", err)
return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
"status": "error",
"message": "Could not create table",
"data": err.Error(),
})
}

return c.Status(fiber.StatusCreated).JSON(fiber.Map{
"status": "success",
"message": "Table created",
"data": table,
})
}
```

****

****

customers_request_models.go_275804e3b59c6ea6c67c479bc18c234c.txt

****

```
package request_models

type CreateCustomerRequest struct {
FirstName string `json:"first_name"`
LastName  string `json:"last_name"`
Email     string `json:"email"`
Phone     string `json:"phone"`
}
```

****

****

database.go_ff5127fb855e44836b9adf45359843cb.txt

****

```
package config

import (
"fmt"
```

```
"gorm.io/driver/postgres"
"gorm.io/gorm"
)

func SetupDatabaseConnection() *gorm.DB {
host := "localhost"
user := "postgres"
dbname := "ServeTech_v1"
sslmode:= "disable"
password := "postgres"

dsn := fmt.Sprintf("host=%s user=%s dbname=%s sslmode=%s password=%s", host, user, dbname, sslr
db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
if err != nil {
panic("Failed to connect to database!")
}
return db
}

func CloseDatabaseConnection(db *gorm.DB) {
dbSQL, err := db.DB()
if err != nil {
panic("Failed to close connection to database!")
}
dbSQL.Close()
}
```

****

****

date_format_checker.go_1c984bf17d1d01e85e38a7808f79c3db.txt

****

```
package utils

import (
"time"
)

// IsDateFormatValid checks if the provided date string is in the format "2006-01-02 15:04:05".
func IsDateFormatValid(dateStr string) bool {
layout := "2006-01-02 15:04:05" // Define the layout based on Go's reference date
_, err := time.Parse(layout, dateStr)
return err == nil // If there's no error, the format is correct
}
```

****

****

db_cancel_order.go_d399a85c76c2a0f4d3f39c5820277534.txt

****

```
package db_orders

import (
"github.com/jimzord12/serve-tech/api/models"
"gorm.io/gorm"
)

func CancelOrderInDatabase(db *gorm.DB, orderID string) (*models.Order, error) {
var order models.Order
result := db.First(&order, orderID)
if result.Error != nil {
return nil, result.Error
}

if order.Status == models.OrderCancelled || order.Status == models.OrderCompleted {
return &order, nil
}

order.Status = models.OrderCancelled
if err := db.Save(&order).Error; err != nil {
return nil, err
}

return &order, nil
}
```

****

****

db_complete_order.go_04547ffae30f4b68a559246d90ded884.txt

****

```
package db_orders

import (
"github.com/jimzord12/serve-tech/api/models"
"gorm.io/gorm"
)

func CompleteOrderInDatabase(db *gorm.DB, orderID string) (*models.Order, error) {
var order models.Order
result := db.First(&order, orderID)
if result.Error != nil {
return nil, result.Error
}

if order.Status == models.OrderCompleted || order.Status == models.OrderCancelled {
```

```
return &order, nil
}

order.Status = models.OrderCompleted
if err := db.Save(&order).Error; err != nil {
return nil, err
}

return &order, nil
}
```

****

****

db_create_history_order.go_bf77fce072db6b672377ad1969343640.txt

****

```
package db_history_orders

import (
"github.com/jimzord12/serve-tech/api/models"
"github.com/jimzord12/serve-tech/api/models/request_models"
"gorm.io/gorm"
)

// CreateHistoryOrderInDatabase inserts a new history order record into the database.
func CreateHistoryOrderInDatabase(db *gorm.DB, request *request_models.CreateHistoryOrderRequest
historyOrder := models.OrderHistory{
OrderID:    int(request.OrderID),
CustomerID: request.CustomerID,
TableID:    request.TableID,
OrderTime:  request.OrderTime,
Status:     request.Status,
Price:      request.Price,
WaiterID:   request.WaiterID,
}

if err := db.Create(&historyOrder).Error; err != nil {
return nil, err
}
return &historyOrder, nil
}
```

****

****

db_create_order.go_3969fa1ac2b0b500a97d258a160bb87d.txt

****

```
package db_orders

import (
"github.com/jimzord12/serve-tech/api/models"
"github.com/jimzord12/serve-tech/api/models/request_models"
"gorm.io/gorm"
)

func CreateOrderInDatabase(db *gorm.DB, orderRequest *request_models.CreateOrderRequest) (*mode

order := models.Order{
TableID:    orderRequest.TableID,
CustomerID: orderRequest.CustomerID,
OrderTime:  orderRequest.OrderTime,
Status:     models.OrderPlaced, // OrderPlaced -> 0
}

if err := db.Create(&order).Error; err != nil {
return nil, err
}

return &order, nil
}

********************************************************************************

********************************************************************************

db_create_table.go_6d5c50215b2d1f3226ce7a00ef3b4ea6.txt

********************************************************************************

package db_tables

import (
"github.com/jimzord12/serve-tech/api/models"
"github.com/jimzord12/serve-tech/api/models/request_models"
"gorm.io/gorm"
)

// CreateTableInDatabase encapsulates the logic to create a new table in the database.
func CreateTableInDatabase(db *gorm.DB, tableRequest *request_models.CreateTableRequest) (*mode
table := models.Table{
Number:     tableRequest.Number,
Capacity:   tableRequest.Capacity,
Status:     models.TableAvailable,
QrCodeHash: tableRequest.QrCodeHash,
}

if err := db.Create(&table).Error; err != nil {
return nil, err
}
```

```
return &table, nil
}
```

****

****

db_customer_exists.go_75d8afa5989a51211b269723e6e92675.txt

****

```
package db_utils

import (
"github.com/jimzord12/serve-tech/api/models" // Adjust the import path based on your actual model locati
"gorm.io/gorm"
)

// DoesCustomerExist checks if a customer exists in the database by their ID.
func DoesCustomerExist(db *gorm.DB, customerID uint) (bool, error) {
var customer models.Customer
result := db.First(&customer, customerID)
if result.Error != nil {
if result.Error == gorm.ErrRecordNotFound {
return false, nil // Customer does not exist
}
return false, result.Error // Some other error occurred
}
return true, nil // Customer exists
}
```

****

****

db_delete_order.go_9fd30828eacc03812bf31d6f6e445ea5.txt

****

```
package db_orders

import (
"fmt"
"strings"

"github.com/jimzord12/serve-tech/api/models"
"gorm.io/gorm"
)

// DeleteOrderInDatabase encapsulates the logic to delete an order in the database.
```

```
func DeleteOrderInDatabase(db *gorm.DB, orderID string) (*models.Order, error) {
var order models.Order
if err := db.First(&order, orderID).Error; err != nil {
return nil, err
}

if err := db.Delete(&order).Error; err != nil {
if strings.Contains(err.Error(), "violates foreign key constraint \"fk_current_order\"") {
return nil, fmt.Errorf("order deletion failed, because Order is still ACTIVE in table [%d]: %w", order.TableI
}
return nil, err
}

return &order, nil
}
```

****

****

db_delete_table.go_cefd3f7d3116e5e916b2516ffd38f64f.txt

****

```
package db_tables

import (
"fmt"
"strings"

"github.com/jimzord12/serve-tech/api/models"
"gorm.io/gorm"
)

// DeleteTableInDatabase encapsulates the logic to delete a table in the database.
func DeleteTableInDatabase(db *gorm.DB, tableID string) (*models.Table, error) {
var table models.Table
if err := db.First(&table, tableID).Error; err != nil {
return nil, err
}

if table.CurrentOrderID != nil && *table.CurrentOrderID != 4 && *table.CurrentOrderID != 5 {
return nil, fmt.Errorf("table has an active order with ID [%d]", *table.CurrentOrderID)
}

if err := db.Delete(&table).Error; err != nil {
if strings.Contains(err.Error(), "violates foreign key constraint \"fk_table\" on table \"orders\"") {
var order models.Order
db.Where("table_id = ?", table.ID).First(&order)
return nil, fmt.Errorf("table with ID [%d] could not be deleted due to foreign key constraint violation in orde
}
return nil, fmt.Errorf("table with ID [%d] could not be deleted", table.ID)
}
```

```
return &table, nil
}
```

****

****

db_get_all_orders.go_c4ce5c29a2008888c725f032dcfb172c.txt

****

```
package db_orders

import (
"github.com/jimzord12/serve-tech/api/models"
"gorm.io/gorm"
)

// GetAllOrdersFromDatabase retrieves all orders from the database.
func GetAllOrdersFromDatabase(db *gorm.DB) ([]models.Order, error) {
var orders []models.Order
if err := db.Find(&orders).Error; err != nil {
return nil, err
}
return orders, nil
}
```

****

****

db_get_all_today_orders.go_4265b7234b9ec5cf8607c5ed33466a80.txt

****

```
package db_orders

import (
"fmt"
"time"

"github.com/jimzord12/serve-tech/api/models"
"gorm.io/gorm"
)

// GetTodayOrdersFromDatabase retrieves all orders created today.
func GetTodayOrdersFromDatabase(db *gorm.DB) ([]models.Order, error) {
var orders []models.Order
today := time.Now().Format("2006-01-02") // Get current date in YYYY-MM-DD format
fmt.Println(today)
```

```
// Use the DATE function to compare only the date part of order_time with today's date
if err := db.Where("DATE(order_time) = ?", today).Find(&orders).Error; err != nil {
return nil, err
}

return orders, nil
}
```

****

****

db_get_business_details.go_019d0f95e8ee3d73bb50b13d211c2511.txt

****

```
package db_business

import (
"github.com/jimzord12/serve-tech/api/models"
"gorm.io/gorm"
)

// GetBusinessDetailsFromDatabase encapsulates the logic to fetch business details from the database.
func GetBusinessDetailsFromDatabase(db *gorm.DB, businessId string) (*models.BusinessDetails, error)
var businessDetails models.BusinessDetails
if err := db.First(&businessDetails, businessId).Error; err != nil {
return nil, err
}
return &businessDetails, nil
}
```

****

****

db_get_order.go_1cde05f85ffe9f949c4e1a5a3143074e.txt

****

```
package db_orders

import (
"github.com/jimzord12/serve-tech/api/models"
"gorm.io/gorm"
)

// GetOrderFromDatabase fetches an order by its ID from the database.
func GetOrderFromDatabase(db *gorm.DB, orderID string) (*models.Order, error) {
var order models.Order
```

```
if err := db.First(&order, orderID).Error; err != nil {
return nil, err
}
return &order, nil
}
```

****

****

db_is_table_available.go_209ec65030b0732e207f5c105367ae0f.txt

****

```
package db_utils

import (
"errors"
"fmt"

"github.com/jimzord12/serve-tech/api/models" // Adjust the import path based on your actual model locati
"gorm.io/gorm"
)

// IsTableAvailable checks if the table exists and is available
func IsTableAvailable(db *gorm.DB, tableID uint) (bool, error) {
var table models.Table
result := db.First(&table, tableID)
if result.Error != nil {
if result.Error == gorm.ErrRecordNotFound {
return false, errors.New("table does not exist in db") // Table does not exist
}
return false, result.Error // Some other error occurred
}

if table.CurrentOrderID != nil {
return false, fmt.Errorf("table [%d] is already occupied with order ID [%d]", table.ID, *table.CurrentOrderID
}

return table.Status == models.TableAvailable, nil // Check if the status is 'Available'
}
```

****

****

db_update_order.go_42ef35007009bd3b8356d1f87f1fd80b.txt

****

```
package db_orders
```

```
import (
"github.com/jimzord12/serve-tech/api/models"
"gorm.io/gorm"
)

// UpdateOrderInDatabase encapsulates the logic to update an order in the database.
func UpdateOrderInDatabase(db *gorm.DB, orderID string, updatedOrderData map[string]interface{}) (*m
var order models.Order
if err := db.First(&order, orderID).Error; err != nil {
return nil, err
}

if err := db.Model(&order).Updates(updatedOrderData).Error; err != nil {
return nil, err
}

return &order, nil
}
```

****

****

db_update_table.go_cb2537ce9002ba66e78e6fcb7ea9956a.txt

****

```
package db_tables

import (
"github.com/jimzord12/serve-tech/api/models"
"gorm.io/gorm"
)

// UpdateTableInDatabase encapsulates the logic to update a table in the database.
func UpdateTableInDatabase(db *gorm.DB, tableID string, updatedTableData map[string]interface{}) (*m
var table models.Table
if err := db.First(&table, tableID).Error; err != nil {
return nil, err
}

if err := db.Model(&table).Updates(updatedTableData).Error; err != nil {
return nil, err
}

return &table, nil
}
```

****

****

delete_order.go_fe5c2c098e287074c637fa472f2f97d5.txt

****

package orders

import (
"strings"

"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_orders"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// DeleteOrder - DELETE handler
func DeleteOrder(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

orderID := c.Params("id")
order, err := db_orders.DeleteOrderInDatabase(db, orderID)
if err != nil {
if strings.Contains(err.Error(), "order deletion failed, because Order is still ACTIVE") {
utils.LogErrorToConsole("Handler: 'DeleteOrder'", err)
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{"status": "error", "message": err.Error(), "data":
}

utils.LogErrorToConsole("Handler: 'DeleteOrder'", err)
return c.Status(fiber.StatusNotFound).JSON(fiber.Map{"status": "error", "message": "Order not found or c
}

return c.JSON(fiber.Map{"status": "success", "message": "Order deleted", "data": order})
}

****

****

delete_table.go_42f00a9ec47dba1bfe8c5b7d0e87c206.txt

****

package tables

import (
"fmt"

"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_tables"
"github.com/jimzord12/serve-tech/config"

```
"github.com/jimzord12/serve-tech/utils"
)

// DeleteTable - DELETE handler
func DeleteTable(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

tableID := c.Params("id")
table, err := db_tables.DeleteTableInDatabase(db, tableID)
if err != nil {
utils.LogErrorToConsole("Handler: 'DeleteTable'", err)
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{
"status": "error",
"message": fmt.Sprintf("Handler: 'DeleteTable', %s", err.Error()),
"data": nil,
})
}

return c.JSON(fiber.Map{"status": "success", "message": "Table deleted", "data": table})
}
```

****

****

enums.go_51bfef3c40491f9ebe9ddb456877f396.txt

****

```
package models

type StaffRole int

const (
WaiterC  StaffRole = iota // 0
WaiterB                   // Automatically increments, assigns 1
WaiterA                   // 2
CashierC                  // 3
CashierB                  // 4
CashierA                  // 5
ChefC                     // 6
ChefB                     // 7
ChefA                     // 8
ManagerC                  // 9
ManagerB                  // 10
ManagerA                  // 11
Admin                     // 12
// Add more roles as needed
)

// - OrderStatus - //
```

```
type OrderStatus int

const (
OrderPlaced    OrderStatus = iota // 0
OrderPreparing                    // Automatically increments, assigns 1
OrderReady                        // 2
OrderServed                       // 3
OrderCancelled                    // 4
OrderCompleted                    // 5
// Add more statuses as needed
)

// - TableStatus - //
type TableStatus int

const (
TableAvailable TableStatus = iota // 0
TableOccupied                     // Automatically increments, assigns 1
TableReserved                     // 2
// Add more statuses as needed
)
```

****

****

error_logger.go_bb585192b637bc1a251e5c0627252ac3.txt

****

```
package utils

import "fmt"

func LogErrorToConsole(message string, err error, ) {
fmt.Println("===> | -ERROR- | " + message)
fmt.Println(err)
}
```

****

****

get_all_orders.go_3fcb18cfcc77ec24e81be91a249ecebf.txt

****

```
package orders

import (
"github.com/gofiber/fiber/v2"
```

```
"github.com/jimzord12/serve-tech/api/db_functions/db_orders"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// GetAllOrders - GET handler
func GetAllOrders(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

orders, err := db_orders.GetAllOrdersFromDatabase(db)
if err != nil {
utils.LogErrorToConsole("Handler: 'GetAllOrders'", err)
return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
"status": "error",
"message": "Failed to retrieve orders",
"data": err,
})
}

return c.JSON(fiber.Map{"status": "success", "message": "retrieved all orders", "data": orders})
}
```

****

****

get_all_today_orders.go_71c61bc23581dbb7b4313add30e89eee.txt

****

```
package orders

import (
"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_orders"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// GetTodayOrders - GET handler
func GetTodayOrders(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

orders, err := db_orders.GetTodayOrdersFromDatabase(db)
if err != nil {
utils.LogErrorToConsole("Handler: 'GetTodayOrders'", err)
return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{
"status": "error",
"message": "Failed to retrieve today's orders",
"data": nil,
```

```
})
}

return c.JSON(fiber.Map{
"status": "success",
"message": "Retrieved today's orders",
"data": orders,
})
}
```

****

****

get_business_details.go_5b1f5168cfed1f23ebce673ff8bb6af1.txt

****

```
package business

import (
"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_business"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// GetBusinessDetails - GET handler
func GetBusinessDetails(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

businessId := c.Params("id")
businessDetails, err := db_business.GetBusinessDetailsFromDatabase(db, businessId)
if err != nil {
utils.LogErrorToConsole("Handler: 'GetBusinessDetails'", err)
return c.Status(fiber.StatusNotFound).JSON(fiber.Map{
"status": "error",
"message": "Retrieving business details failed!",
"data": err,
})
}

return c.JSON(businessDetails)
}
```

****

****

get_order.go_8b9e1ba009c9ee1a7ec88e08db79a891.txt

```
********************************************************************************

package orders

import (
"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_orders"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// GetOrder - GET handler
func GetOrder(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

orderID := c.Params("id")
order, err := db_orders.GetOrderFromDatabase(db, orderID)
if err != nil {
utils.LogErrorToConsole("Handler: 'GetOrder'", err)
return c.Status(fiber.StatusNotFound).JSON(fiber.Map{
"status": "error",
"message": "Order not found",
"data": err,
})
}

return c.JSON(fiber.Map{"status": "success", "message": "retrieved order", "data": order})
}


********************************************************************************

********************************************************************************

go.mod_14855daf6e249c7c25ebfb22600b13ed.txt

********************************************************************************

module github.com/jimzord12/serve-tech

go 1.22.1

require gorm.io/driver/postgres v1.5.7

require (
github.com/andybalholm/brotli v1.0.5 // indirect
github.com/google/uuid v1.5.0 // indirect
github.com/klauspost/compress v1.17.0 // indirect
github.com/mattn/go-colorable v0.1.13 // indirect
github.com/mattn/go-isatty v0.0.20 // indirect
github.com/mattn/go-runewidth v0.0.15 // indirect
```

github.com/rivo/uniseg v0.2.0 // indirect
github.com/valyala/bytebufferpool v1.0.0 // indirect
github.com/valyala/fasthttp v1.51.0 // indirect
github.com/valyala/tcplisten v1.0.0 // indirect
golang.org/x/sys v0.15.0 // indirect
)

require (
github.com/gofiber/fiber/v2 v2.52.2
github.com/jackc/pgpassfile v1.0.0 // indirect
github.com/jackc/pgservicefile v0.0.0-20221227161230-091c0ba34f0a // indirect
github.com/jackc/pgx/v5 v5.4.3 // indirect
github.com/jinzhu/inflection v1.0.0 // indirect
github.com/jinzhu/now v1.1.5 // indirect
golang.org/x/crypto v0.14.0 // indirect
golang.org/x/text v0.13.0 // indirect
gorm.io/gorm v1.25.8 // indirect
)

****

****

go.sum_f051fff3514252aa37a19cb6ae399886.txt

****

github.com/andybalholm/brotli v1.0.5 h1:8uQZIdzKmjc/iuPu7O2ioW48L81FgatrcpfFmiq/cCs=
github.com/andybalholm/brotli v1.0.5/go.mod h1:fO7iG3H7G2nSZ7m0zPUDn85XEX2GTukHGRSepvi9E
github.com/davecgh/go-spew v1.1.0/go.mod h1:J7Y8YcW2NihsgmVo/mv3lAwl/skON4iLHjSsI+c5H38=
github.com/gofiber/fiber/v2 v2.52.2 h1:b0rYH6b06Df+4NyrbdptQL8ifuxw/Tf2DgfkZkDaxEo=
github.com/gofiber/fiber/v2 v2.52.2/go.mod h1:KEOE+cXMhXG0zHc9d8+E38hoX+ZN7bhOtgeF2oT6jrQ=
github.com/google/uuid v1.5.0 h1:1p67kYwdtXjb0gL0BPiP1Av9wiZPo5A8z2cWkTZ+eyU=
github.com/google/uuid v1.5.0/go.mod h1:TIyPZe4MgqvfeYDBFedMoGGpEw/LqOeaOT+nhxU+yHo=
github.com/jackc/pgpassfile v1.0.0 h1:/6Hmqy13Ss2zCq62VdNG8tM1wchn8zjSGOBJ6icpsIM=
github.com/jackc/pgpassfile v1.0.0/go.mod h1:CEx0iS5ambNFdcRtxPj5JhEz+xB6uRky5eyVu/W2HEg=
github.com/jackc/pgservicefile v0.0.0-20221227161230-091c0ba34f0a h1:bbPeKD0xmW/Y25WS6cokEs
github.com/jackc/pgservicefile v0.0.0-20221227161230-091c0ba34f0a/go.mod h1:5TJZWKEWniPve33vl
github.com/jackc/pgx/v5 v5.4.3 h1:cxFyXhxlvAifxnkKKdlxv8XqUf59tDlYjnV5YYfsJJY=
github.com/jackc/pgx/v5 v5.4.3/go.mod h1:Ig06C2Vu0t5qXC60W8sqIthScaEnFvojjj9dSljmHRA=
github.com/jinzhu/inflection v1.0.0 h1:K317FqzuhWc8YvSVlFMCCUb36O/S9MCKRDI7QkRKD/E=
github.com/jinzhu/inflection v1.0.0/go.mod h1:h+uFLlag+Qp1Va5pdKtLDYj+kHp5pxUVkryuEj+Srlc=
github.com/jinzhu/now v1.1.5 h1:/o9tlHleP7gOFmsnYNz3RGnqzefHA47wQpKrrdTIwXQ=
github.com/jinzhu/now v1.1.5/go.mod h1:d3SSVoowX0Lcu0IBviAWJpolVfI5UJVZZ7cO71lE/z8=
github.com/klauspost/compress v1.17.0 h1:Rnbp4K9EjcDuVuHtd0dgA4qNuv9yKDYKK1ulpJwgrqM=
github.com/klauspost/compress v1.17.0/go.mod h1:ntbaceVETuRiXiv4DpjP66DpAtAGkEQskQzEyD//IeE
github.com/mattn/go-colorable v0.1.13 h1:fFA4WZxdEF4tXPZVKMLwD8oUnCTTo08duU7wxecdEvA=
github.com/mattn/go-colorable v0.1.13/go.mod h1:7S9/ev0klgBDR4GtXTXX8a3vIGJpMovkB8vQcUbaXH
github.com/mattn/go-isatty v0.0.16/go.mod h1:kYGgaQfpe5nmfYZH+SKPsOc2e4SrIfOl2e/yFXSvRLM=
github.com/mattn/go-isatty v0.0.20 h1:xfD0iDuEKnDkl03q4limB+vH+GxLEtL/jb4xVJSWWEY=
github.com/mattn/go-isatty v0.0.20/go.mod h1:W+V8PltTTMOvKvAeJH7IuucS94S2C6jfK/D7dTCTo3Y=
github.com/mattn/go-runewidth v0.0.15 h1:UNAjwbU9l54TA3KzvqLGxwWjHmMgBUVhBiTjelZgg3U=

github.com/mattn/go-runewidth v0.0.15/go.mod h1:Jdepj2loyihRzMpdS35Xk/zdY8IAYHsh153qUoGf23w=
github.com/pmezard/go-difflib v1.0.0/go.mod h1:iKH77koFhYxTK1pcRnkKkqfTogsbg7gZNVY4sRDYZ/4=
github.com/rivo/uniseg v0.2.0 h1:S1pD9weZBuJdFmowNwbpi7BJ8TNftyUImj/0WQi72jY=
github.com/rivo/uniseg v0.2.0/go.mod h1:J6wj4VEh+S6ZtnVlnTBMWIodfgj8LQOQFoIToxlJtxc=
github.com/stretchr/objx v0.1.0/go.mod h1:HFkY916IF+rwdDfMAkV7OtwuqBVzrE8GR6GFx+wExME=
github.com/stretchr/testify v1.3.0/go.mod h1:M5WIy9Dh21IEIfnGCwXGc5bZfKNJtfHm1UVUgZn+9EI=
github.com/stretchr/testify v1.7.0/go.mod h1:6Fq8oRcR53rry900zMqJjRRixrwX3KX962/h/Wwjteg=
github.com/valyala/bytebufferpool v1.0.0 h1:GqA5TC/0021Y/b9FG4Oi9Mr3q7XYx6KllzawFIhcdPw=
github.com/valyala/bytebufferpool v1.0.0/go.mod h1:6bBcMArwyJ5K/AmCkWv1jt77kVWyCJ6HpOuEn7z0
github.com/valyala/fasthttp v1.51.0 h1:8b30A5JlZ6C7AS81RsWjYMQmrZG6feChmgAolCl1SqA=
github.com/valyala/fasthttp v1.51.0/go.mod h1:oI2XroL+lI7vdXyYoQk03bXBThfFl2cVdIA3Xl7cH8g=
github.com/valyala/tcplisten v1.0.0 h1:rBHj/Xf+E1tRGZyWIWwJDiRY0zc1Js+CV5DqwacVSA8=
github.com/valyala/tcplisten v1.0.0/go.mod h1:T0xQ8SeCZGxckz9qRXTfG43PvQ/mcWh7FwZEA7Ioqkc=
golang.org/x/crypto v0.14.0 h1:wBqGXzWJW6m1XrIKlAH0Hs1JJ7+9KBwnIO8v66Q9cHc=
golang.org/x/crypto v0.14.0/go.mod h1:MVFd36DqK4CsrnJYDkBA3VC4m2GkXAM0PvzMCn4JQf4=
golang.org/x/sys v0.0.0-20220811171246-fbc7d0a398ab/go.mod h1:oPkhp1MJrh7nUepCBck5+mAzfO9J
golang.org/x/sys v0.6.0/go.mod h1:oPkhp1MJrh7nUepCBck5+mAzfO9JrbApNNgaTdGDITg=
golang.org/x/sys v0.15.0 h1:h48lPFYpsTvQJZF4EKyI4aLHaev3CxivZmv7yZig9pc=
golang.org/x/sys v0.15.0/go.mod h1:/VUhepiaJMQUp4+oa/7Zr1D23ma6VTLIYjOOTFZPUcA=
golang.org/x/text v0.13.0 h1:ablQoSUd0tRdKxZewP80B+BaqeKJuVhuRxj/dkrun3k=
golang.org/x/text v0.13.0/go.mod h1:TvPlkZtksWOMsz7fbANvkp4WM8x/WCo/om8BMLbz+aE=
gopkg.in/check.v1 v0.0.0-20161208181325-20d25e280405/go.mod h1:Co6ibVJAznAaIkqp8huTwlJQCZ0
gopkg.in/yaml.v3 v3.0.0-20200313102051-9f266ea9e77c/go.mod h1:K4uyk7z7BCEPqu6E+C64Yfv1cQ7
gorm.io/driver/postgres v1.5.7 h1:8ptbNJTDbEmhdr62uReG5BGkdQyeasu/FZHxI0IMGnM=
gorm.io/driver/postgres v1.5.7/go.mod h1:3e019WlBaYI5o5LIdNV+LyxCMNtLOQETBXL2h4chKpA=
gorm.io/gorm v1.25.8 h1:WAGEZ/aEcznN4D03laj8DKnehe1e9gYQAjW8xyPRdeo=
gorm.io/gorm v1.25.8/go.mod h1:hbnx/Oo0ChWMn1BIhpy1oYozzpM15i4YPuHDmfYtwg8=

****

****

history_orders_rquest_models.go_0f68a7991c5633a89faca4e18094eac1.txt

****

```
package request_models

import "github.com/jimzord12/serve-tech/api/models"

// import "github.com/jimzord12/serve-tech/api/models"

type CreateHistoryOrderRequest struct {
CustomerID uint              `json:"customer_id"`
OrderTime  string            `json:"order_time"`
WaiterID   uint              `json:"waiter_id"`
Status     models.OrderStatus `json:"status"`
TableID    uint              `json:"table_id"`
Price      float64           `json:"price"`
OrderID■■uint                `json:"order_id"`

// Include any other fields that should be set during order creation
```

```
}

********************************************************************************

********************************************************************************

id_caster_to_uint.go_748aac24465e6bb391b2d9957ac058c5.txt

********************************************************************************

package utils

import (
"errors"
"strconv"
)

func StringToInt(sValue string) (int, error) {
// Convert tableID from string to uint
parsedValue, err := strconv.ParseInt(sValue, 10, 64)
if err != nil {
return -1, errors.New("failed to parse string to uint")
}
return int(parsedValue), nil
}

********************************************************************************

********************************************************************************

main.go_886a6a41a2ef2e74e3b978c8b52140af.txt

********************************************************************************

package main

import (
"github.com/jimzord12/serve-tech/api/handlers/business"
"github.com/jimzord12/serve-tech/api/handlers/customers"
"github.com/jimzord12/serve-tech/api/handlers/history_orders"
"github.com/jimzord12/serve-tech/api/handlers/orders"
"github.com/jimzord12/serve-tech/api/handlers/tables"

"github.com/gofiber/fiber/v2"
)

// Meaning of the emojis:
// ■ - Needs testing
// ■ - Tested and working
// ■ - In progress
// ■ - Needs documentation
// ■ - Documented
// ■ - Needs fixing
```

```
// ■ - Important
// ■ - To-do
// ■ - Needs refactoring
// ■ - Needs focus
// ■ - Ready for deployment

func main() {
app := fiber.New()

app.Get("/", func(c *fiber.Ctx) error {
return c.SendString("Welcome To Serve Tech!")
})
// Business
app.Get("/businessdetails", business.GetBusinessDetails)

// Orders
app.Post("/create-order", orders.CreateOrder) ■■■// ■ [■] - ■
app.Put("/update-order/:id", orders.UpdateOrder) ■■// ■ [■] - ■
app.Put("/cancel-order/:id", orders.CancelOrder) ■■// ■ [■] - ■
app.Put("/complete-order/:id", orders.CompleteOrder) ■// ■ [■] - ■
app.Get("/get-order/:id", orders.GetOrder)■■■■// ■ [■] - ■
app.Get("/get-all-orders", orders.GetAllOrders)■■■// ■ [■] - ■
app.Get("/get-all-today-orders", orders.GetTodayOrders) // ■ [■] - ■
app.Delete("/delete-order/:id", orders.DeleteOrder)■■// ■ [■] - ■

// Tables
app.Post("/create-table", tables.CreateTable)■■■// ■ [■] - ■
app.Put("/update-table/:id", tables.UpdateTable)■■// ■ [■] - ■
app.Delete("/delete-table/:id", tables.DeleteTable)■■// ■ [■] - ■

// Customers
app.Post("/create-customer", customers.CreateCustomer)■// ■ [■] - ■

// History Orders
app.Post("/create-history-order", history_orders.CreateHistoryOrder) // ■ [■] - ■


app.Listen(":3000")
}


********************************************************************************

********************************************************************************

methods.go_ea9bc697343fdba13536ce50b512129c.txt

********************************************************************************

package models

import "fmt"
```

```
// STAFF - A method that returns the full name of a staff member
func (s *Staff) FullName() string {
return fmt.Sprintf("%s %s", s.FirstName, s.LastName)
}

// STAFF - A method to check if a staff member has a certain role
func (s *Staff) IsRole(role StaffRole) bool {
return s.Role == role
}
```

****

****

models.go_2c3abf7a7ee7bc2bf943383ed3609b66.txt

****

```
package models

type BusinessDetails struct {
ID          uint `gorm:"primaryKey"`
Name        string
Address     string
TotalTables int
OpeningTime string
ClosingTime string
}

type Order struct {
ID         uint        `gorm:"primaryKey"`
TableID    uint        `gorm:"not null"`
CustomerID uint        `gorm:"not null"`
OrderTime  string      `gorm:"not null"`
Status     OrderStatus `gorm:"not null"`
}

type OrderItem struct {
ID        uint    `gorm:"primaryKey"`
OrderID   uint    `gorm:"not null"`
ProductID uint    `gorm:"not null"`
Quantity  int     `gorm:"not null"`
Notes     string  // Additional notes or special requests for the item
Order     Order   `gorm:"foreignKey:OrderID"`
Product   Product `gorm:"foreignKey:ProductID"`
}

type OrderHistory struct {
ID         uint        `gorm:"primaryKey"`
OrderID    int         `gorm:"not null"`
CustomerID uint        `gorm:"not null"`
TableID    uint        `gorm:"not null"`
OrderTime  string      `gorm:"not null"`
```

```
Status     OrderStatus `gorm:"not null"`
Price      float64     `gorm:"not null"`
WaiterID   uint        `gorm:"not null"`
}

type Table struct {
ID            uint        `gorm:"primaryKey"`
Number        int         `gorm:"unique;not null"` // Table number or identifier
Capacity      int         `gorm:"not null"`        // How many people can sit at the table
Status        TableStatus // e.g., Available = 0, Occupied = 1, Reserved = 2
CurrentOrderID *uint      // Link to an active order, if any. Pointer to allow nil (NULL).
QrCodeHash    string      // Hash of the table's QR code
}

type Staff struct {
ID          uint      `gorm:"primaryKey"`
FirstName   string    `gorm:"not null"`
LastName    string    `gorm:"not null"`
Role        StaffRole `gorm:"not null"`
Email       string    `gorm:"unique"`
Phone       string
AccessLevel uint `gorm:"default:0"` // Access level, defaults to 0
// Other relevant fields...
}

type Product struct {
ID          uint    `gorm:"primaryKey"`
ProductCode uint    `gorm:"unique;not null"`
Name        string  `gorm:"not null"`
Description string  // Detailed description of the product
Price       float64 `gorm:"not null"`
Stock       int     // Quantity available; useful if tracking inventory
}

type Customer struct {
ID        uint   `gorm:"primaryKey"`
FirstName string `gorm:"not null"`
LastName  string `gorm:"not null"`
Email     string `gorm:"unique"`
Phone     string
// Additional fields like loyalty points, address, etc., can be added if needed
}
```

****

****

orders_request_models.go_e6e750379041c1c109583781aa3afb56.txt

****

```
package request_models
```

```
import "github.com/jimzord12/serve-tech/api/models"

type CreateOrderRequest struct {
TableID    uint              `json:"table_id"`
CustomerID uint              `json:"customer_id"`
OrderTime  string            `json:"order_time"`
Status     models.OrderStatus `json:"status"`
// Include any other fields that should be set during order creation
}
```

****

****

tables_request_models.go_8a6371a02342f7c4411c30d1677a6439.txt

****

```
package request_models

type CreateTableRequest struct {
Number     int    `json:"number"`
Capacity   int    `json:"capacity"`
QrCodeHash string `json:"qr_code_hash"`
}

type UpdateTableRequest struct {
Number         int    `json:"number"`
Capacity       int    `json:"capacity"`
QrCodeHash     string `json:"qr_code_hash"`
Status         int    `json:"status"`
CurrentOrderID uint   `json:"current_order_id"`
}
```

****

****

TODOS.md_62b954beb411c86c0e6b333c25cc9034.txt

****

TODOS

For Next Time

1. Test & Perform Error Handling on all the routes.

General

1. When Assigning an ORDER to a TABLE, check if the TABLE already has an ORDER.

- If it does, throw an Error

2. Figure out how to DELETE staff, because the foreign key contraints make it difficult.

Completed

- When Assigning an ORDER to a TABLE (When creating an Order), check everything (table, customer,

****

****

update_order.go_409414c86903df43e5517ecaafad394e.txt

****

```
package orders

import (
"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_orders"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// UpdateOrder - PUT handler
func UpdateOrder(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

orderID := c.Params("id")

var updatedOrderData map[string]interface{}
if err := c.BodyParser(&updatedOrderData); err != nil {
utils.LogErrorToConsole("Handler: 'UpdateOrder', Cannot parse JSON", err)
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{"status": "error", "message": "Bad request", "da
}

order, err := db_orders.UpdateOrderInDatabase(db, orderID, updatedOrderData)
if err != nil {
utils.LogErrorToConsole("Handler: 'UpdateOrder'", err)
return c.Status(fiber.StatusNotFound).JSON(fiber.Map{"status": "error", "message": "Order not found or c
}

return c.JSON(fiber.Map{"status": "success", "message": "Order updated", "data": order})
}
```

****

****

update_table.go_5ef9fced50110b8991305cd6366420a0.txt

```
********************************************************************************

package tables

import (
"github.com/gofiber/fiber/v2"
"github.com/jimzord12/serve-tech/api/db_functions/db_tables"
"github.com/jimzord12/serve-tech/config"
"github.com/jimzord12/serve-tech/utils"
)

// UpdateTable - PUT handler
func UpdateTable(c *fiber.Ctx) error {
db := config.SetupDatabaseConnection()
defer config.CloseDatabaseConnection(db)

tableID := c.Params("id")

var updatedTableData map[string]interface{}
if err := c.BodyParser(&updatedTableData); err != nil {
utils.LogErrorToConsole("Handler: 'UpdateTable', Cannot parse JSON", err)
return c.Status(fiber.StatusBadRequest).JSON(fiber.Map{"status": "error", "message": "Bad request", "da
}

table, err := db_tables.UpdateTableInDatabase(db, tableID, updatedTableData)
if err != nil {
utils.LogErrorToConsole("Handler: 'UpdateTable'", err)
return c.Status(fiber.StatusInternalServerError).JSON(fiber.Map{"status": "error", "message": "Failed to u
}

return c.JSON(fiber.Map{"status": "success", "message": "Table updated", "data": table})
}


********************************************************************************

********************************************************************************

_db_delete_history_order.go_edf7cc24907d78dd02fe700437315d4a.txt

********************************************************************************

package db_history_orders


********************************************************************************

********************************************************************************

_db_get_all_history_orders.go_edf7cc24907d78dd02fe700437315d4a.txt

********************************************************************************
```

package db_history_orders

****

****

_db_get_history_order.go_edf7cc24907d78dd02fe700437315d4a.txt

****

package db_history_orders

****

****

_db_update_history_order.go_edf7cc24907d78dd02fe700437315d4a.txt

****

package db_history_orders

****

****

_delete_history_order.go_217de1ba03c93f83f7282ac1ccdc6d4f.txt

****

package history_orders

****

****

_get_all_history_orders.go_217de1ba03c93f83f7282ac1ccdc6d4f.txt

****

package history_orders

****

****

_get_history_order.go_217de1ba03c93f83f7282ac1ccdc6d4f.txt

****

package history_orders

****

****

_update_history_order.go_217de1ba03c93f83f7282ac1ccdc6d4f.txt

****

package history_orders

****